Implementation of Internal Model Controller for System with Large Dead Time

By

David Anders Tingdahl

Diploma Thesis for Degree

Erasmus Mundus Space Master:
Master of Space Science and Technology

At

Czech Technical University in Prague,
Faculty of Electrical Engineering,
Department of Control Engineering

Collaborators:

Shanghai Jiao Tong University, Department of Control Engineering and Network Technology

And

Luleå University of Technology, Department of Space Science, Kiruna Space Campus

May 2007

DECLARATION OF INDEPENDENT WORK

I, DAVID ANDERS TINGDAHL, hereby declare that this research project submitted for the degree MASTER OF SPACE SCIENCE AND TECHNOLOGY, is my own independent work that has not been submitted before to any institution by me or anyone else as part of any qualification.

.....

DAVID TINGDAHL

2007-05-22

DATE

Czech Technical University in Prague, Faculty of Electrical Engineering

Department of Control Engineering *Academic Year:* 2006/2007

MASTER THESIS ASSIGNMENT

Student: David Anders Tingdahl

Field of Study: Cybernetics and Measurement - SpaceMaster

Thesis Title: Design and implementation of a networked control algorithm for big time delay system

Thesis Guidelines:

Shanghai JiaoTong University (SJTU) is involved in the Alpha-Magnetic Spectrometer (AMS) project - a particle detector which is supposed to be placed on the international space station in 2009. The main part of the AMS is a 2000 kg superconducting magnet which is to be cooled down to some few degrees Kelvin. The control department of SJTU is involved in the ground support cryogenic equipment, which involves control and monitoring of the cooling down process. Several network topologies are used here, namely Canbus, Profinet and Ethernet, to interface a number of surveillance and control computers to the AMS via a PLC.

Cooling down the magnet is a complex and time consuming process which takes several weeks to accomplish, thus the control algorithms has to take a very large time delay into account. The proposed thesis work is to research on this algorithm, accounting both for the large time delay and also for possible packet losses and delays due to the different network topologies. The algorithm is to be implemented using a Siemens S7-400 PLC which connects to the cooling equipment via PROFIBUS. Since the real cooling equipment is not presently at hand, the algorithm will be tested using a valve / heater configuration.

Bibliography:

Time-delay systems: an overview of some recent advances and open problems, Jean-Pierre Richard, Ecole Centrale de Lille, LAIL (CNRS UMR 8021), BP 48, 59651 Villeneuve d'Ascq, Cedex, France

A Modified Smith Predictor for Controlling a Proces with an Integrator and Long Dead-Time, M. R. Matauiek and A. D. Micii: IEEE TRANSACTIONS ON AUTOMATIC CONTROL, VOL. 41, NO. 8, AUGUST 1996

Autotuning PID Control for Large Time-Delay Processes and Its Application to Paper Basis Weight Control, Wei Tang* and Song-jiao Shi, Automation Department, Shanghai Jiaotong University, Shanghai, P.R. China 200030

Network Based Control Systems: A tutorial, Mo Yuen-Chow and Yodyium Tipsuwan, IECON'01, The 27th Annual Conference of the IEEE Industrial Electronics Society

Supervisor: Ing. Martin Hromčík, Ph.D.
Andreas Johansson

Assignment Date: May 2007

Thesis Due: June 2007

L.S.

prof. Ing. Michael Šebek, DrSc.
Department Head

prof. Ing. Zbyněk Škvor, CSc.
Dean

In Prague 2007-02-21

Abstract

This project is related to the cooling system of the Alpha Magnetic Spectrometer superconducting magnet which involves a big time delay. Internal Model Control is a proven method for systems with large time delays and a control structure using this method is suggested. Since the time delay can be varying during operation, an adaptive algorithm is introduced for tracking its changes. Simulations in MATLAB as well as in a custom made VISUAL C++ program shows its usefulness but further tuning and modification might be necessary before practical implementation.

Table of Contents

Chapter 1: Introduction

This project is derived from Shanghai Jiao Tong University's part in the Alpha Magnetic Spectrometer project. A brief introduction to the AMS and the cooling system of its magnet is given in this chapter.

The Alpha Magnetic Spectrometer project

1.1 Overview

The Alpha Magnetic Spectrometer (AMS) is a particle detector designed to search for anti-matter, dark matter and the origin of cosmic rays in space. [1]. Initiated by CERN in Switzerland, the project involves 16 countries, including China, Russia and USA. In 1998, a prior test model (AMS-01) was successfully evaluated in space for ten days on board the Space Shuttle and using the experimental results, a refined model, the AMS-02, was developed. The AMS-02 is currently under construction and is scheduled to be launched in 2009 from the Kennedy Space Center in USA and thereafter deployed at the International Space Station with an expected operational time of 3 years. To increase the sensitivity of the AMS-02, it was decided to upgrade the existing permanent magnet to to a superconducting system. The new system features a 2300 kg superconducting magnet, currently under construction by Space Cryomagnets Ltd in England. The 14 coils are wired from a high purity aluminum-stabilized mono-strand NbTi conductor and are to be cooled down to 1.8 K using superfluid helium, generating a magnetic field of 0.87 T in its center. Cryogenic magnets has been flown in space before but the AMS-02 magnet will be the by far largest magnet in space until now.

1.2 Cryogenic System

The Cryogenic Ground Support Equipment will be used for cryogenic operations of the magnet on the ground, such as cooling, warming, filling with helium etc. The system is going to be used at various locations (integration sites, space simulator, launch pad) and the versatile design allow for this. As an example, all equipment has to be explosion proof as it is to be operated at the launch pad at Kennedy Space Center. The magnet is to be launched in cold state and the last top-of of superfluid helium will take place only hours before the launch.

The Cryogenic Ground Support Equipment consists of two main parts: the Mechanical System with Dewars, vacuum pumps, valves, etc and the Electrical System which is used for control and monitoring of the processes using a number of networked computers and PLCs. The two parts are currently under construction at Shanghai Jiao Tong University and are scheduled for completion in the last quarter of 2007.

Mechanical system

The Cryogenic Ground Support Mechanical System can be seen in Figure 1

1.1.1 Cooling down the magnet

The cooling down of the magnet coils to 1.8 K is a complex procedure which is performed in three stages as described here.

300K – 80K

The 300K – 80K cryogenic system can be seen in the bottom left part of Figure 1. It connects to the magnet through the valve box via the cryogenic pipes where gaseous helium is circulated. To cool down the magnet, the temperature of the helium is adjusted, using three valves. Three valves are used to adjust the temperature of the flow (measured at the output of the 300K-80K cooling system)

- V1 regulates the cold flow, generated by passing the gaseous helium trough a liquid nitrogen bath at 77K.
- V2 gives access to a intermediate temperature, obtained by passing the output temperature from the magnet through a heat exchanger.
- V3 regulates the flow of helium temperated at 300K (room temperature). This valve is only used for warming the magnet.

To avoid thermal stresses in the material, the maximum allowed temperature difference between in- and outflow during this stage is 50 K. The flow rate of the helium can vary during operation.

90K - 4.2K

In this range, liquid helium is used. A pre-cooled filter is filtering the gas such that all gases except helium becomes solid and is stopped by the filter.

4.2K - 1.8K

Using a vacuum pump, superfluid helium is produced to reach the operating temperature of 1.8 K

Task Statement

Since the 300K-80K system is located away from the AMS magnet, there will be a big time delay introduced in the cooling pipes. Also inside the magnet itself, the cooling pipes will create additional time delay. The length of the time delay can be varying during the operation of the cooling system as the helium flow rate change. Due to the constraint on the temperature different between in- and outflow, it is of a great importance to be able to accurately control the temperature at the outflow from the magnet. Internal Model Control (IMC) is a method which can account for time delays. Research on IMC has previously been carried out at the Shanghai Jiao Tong University and therefore it was decided to use this structure for control. The following task statement is derived:

"To devise a control algorithm using Internal Model Control for systems with large dead time in general and for the AMS-02 magnet 300K – 80 K cooling system in typical. An adaptive algorithm is to be applied to account for the changes in time delay. The control method will be evaluation in the AMS-02 cooling system network available in the control systems lab, controlling a heater / valve configuration."

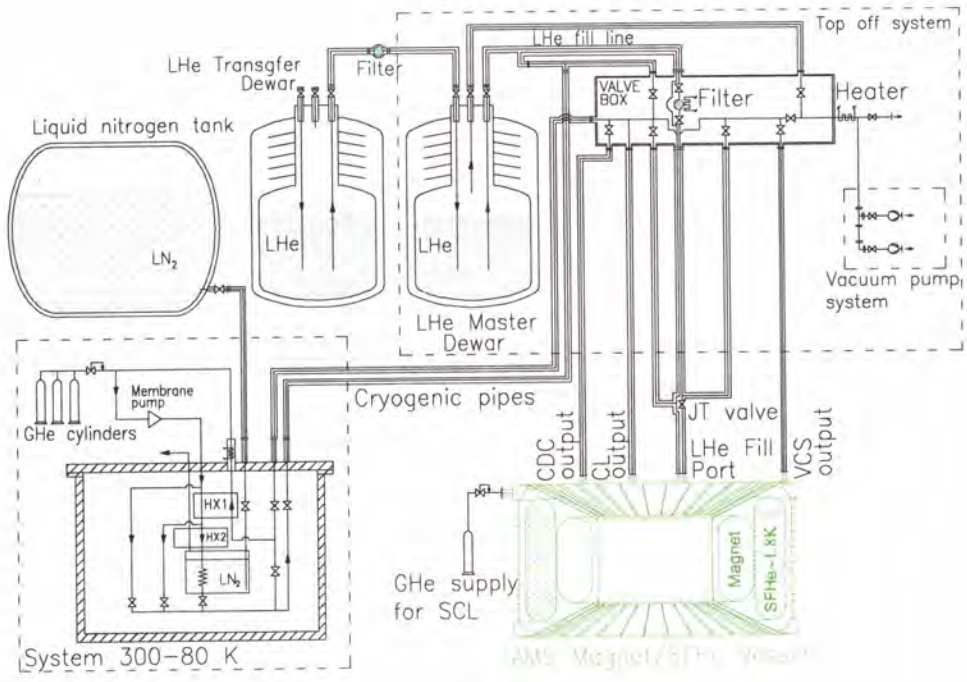

Figure 1: Overview of Cryogenic Ground Support Mechanical System

Layout of the paper

In the two following chapters, basic theory regarding time delay systems and Internal Model Control are discussed. Chapter 4 suggestion for a recursive algorithm that can detect the system's rational parameters as well as the changing time-delay. Chapter 5 is devised to explain the model used and the following Internal Model Control structure. To simulate this model, a Visual C++ program was created which is presented in Chapter 6. The last chapter holds the conclusion and final words.

Chapter 2: Delay systems

A brief introduction to system with time delay is given in this chapter. General properties are discussed as well as the Padé approximation approach.

General properties

Delay time is present in every real-time system since no calculation or action will happen instantly. However in many cases the delay is of such a small magnitude as compared to the process time so that it can be neglected (for example computational time in a digital controller).

The Laplace transform of a delay of L seconds is e^{-Ls}. Considering that the Taylor expansion of e^{-Ls} has an infinite number of terms, it is clear that a (continuous) feedback system with a delay element will have an infinite number of poles and thus an infinite number of states. The Smith Predictor and Internal Model Control (as discussed in this paper) are two methods to deal with this situation. The approach is similar in both cases – the closed loop system is constructed in such a way that the delay element is (ideally) removed from the characteristic equation of the system.

The delay will induce a phase shift of $-Lj\omega$ radians where ω is the angular frequency given in rad/s. To preserve the number of circles around -1 in the Nyqvist plot (i.e: preserve stability), $-Lj\omega$ therefore has to be smaller than the phase margin of the system without delay. A delay will thus directly inflict the maximum allowed bandwidth of the system. For acceptable control performance it is desired to keep the cross-over frequency below 1/L approximately.

Padé approximations

Time delays are often approximated with a rational Padé approximation (from the French mathematician Henri Eugéne Padé). It uses a rational expression for approximating a function (in contrast to for example a Taylor approximation). The most commonly used Padé approximate for e^{-Ls} is the 1/1 approximations where 1/1 stands for the order of numerator and denominator respectively. It is given as:

$$e^{-Ls} \approx \frac{2-Ls}{2+Ls} \tag{1}$$

Comparison betweens a time delay and its Padé approximation can be seen in Figure 2. The beauty of the Padé approximation lies in the frequency domain where the curves can be seen to follow each other closely for low frequencies.

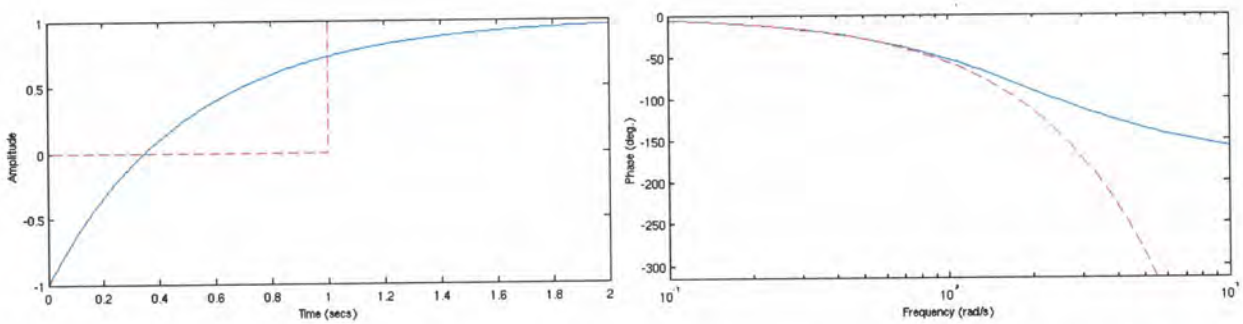

Figure 2: Time and frequency responses of a 1 second time delay (red) along with its 1/1 Padé approximation (blue)

Chapter 3: Internal Model Control

This chapter contains information about Internal Model Control. The structure is derived and its properties are discussed.

Introduction

Internal Model Control (IMC) is a control strategy first proposed by Manfred Morari in 1982 [2]. As the name suggests, the method is based on the idiom that control can be achieved only if the control system encapsulates some representation of the process to be controlled. In the IMC structure this is evident as a complete model of the plant appearing in the block diagram (see Figure 3).

Some of the basic properties of the IMC controller is listed here, derivations can be found in the next section.

- If the model the plant is exactly known, the system behaves as an open loop and the stability issue is trivial.
- Perfect controller (input to output transfer function = 1) can be achieved.
- Zero offset to step input. inherent integral action can be achieved without additional tuning parameters.
- The basic IMC structure has only one tuning parameter and will thus be easy to tune as compared to for example the PID structure.

3.1 Modifications of the IMC structure

For practical implementation, the IMC structure can be converted into PID structure, using Padé approximations for the time delay [3]. Thus it can benefit from all the development available in the PID framework.

The basic formulation assumes a stable plant but control of unstable plants can be achieved using slight modification of the structures. The method described in [4] is shown how to stabilize the plant to maintain robust servo capacities for step inputs and [5] suggests a method where setpoint tracking and disturbance rejection can be designed separately.

It has also been shown that robustness and disturbance rejection can be improved by introducing robust and disturbance compensators to the model and feedback paths respectively [6]. This is especially useful if the system has slow modes which will make a disturbance harmful for a long time. The resulting control design features a 3-degree of freedom sensitivity function.

As explained in the next section, one of the biggest obstacles when designing an IMC structure is to find a suitable inverse of the plant. A non-minimum phase plant will render a unstable and/or non-casual controller which is not realizable in practice, thus there is of great importance to find a suitable approximation to the inverse. The original publication by Morari [2] suggests a predictive algorithm

České vysoké učení technické v Praze
FAKULTA ELEKTROTECHNICKÁ
Katedra řídicí techniky
121 35 Praha 2, Karlovo nám.13
Tel.: 224922372, Fax: 224918646

based on a impulse response representation of the plant. More recent approaches includes inverse approximation using recursive least squares [7] and adaptive algorithms [8].

Internal Model Control Design

3.1 Basic ideas

Consider the stable plant $G(s)$. The so called Perfect Controller suggests the use of an open loop controller $K_{perfect}(s) = G^{-1}(s)$ in series with $G(s)$ to produce a perfect response to any input (reference to output transfer function will be equal to 1). Of course, this open loop control system is not realizable in practice since it cannot account for model mismatch and external disturbances. In addition, if $G(s)$ is non-minimum phase, its inverse will render an unstable or non-casual controller. One of the main advantages of the IMC structure is that it can combine the performance obtained with a Perfect Controller with the robustness of a feedback system as explained below. The basic Internal Model Control structure can be seen in Figure 3 where $Q(s)$ is the IMC controller and $G_M(s)$ is the model of the plant $G(s)$.

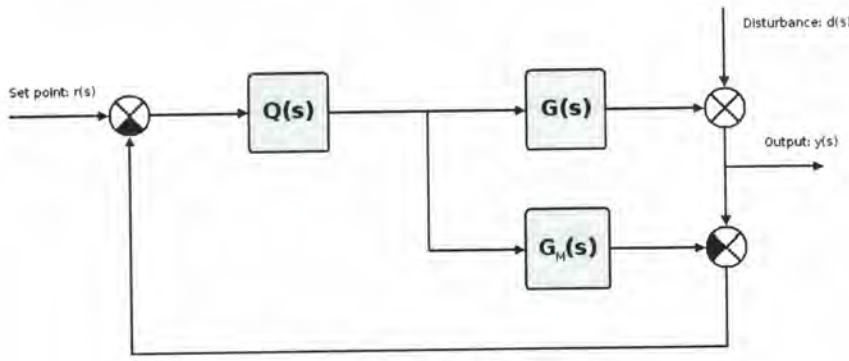

Figure 3: Basic IMC structure

From the block diagram the expression for the output can be derived as:

$$y(s) = \underbrace{\frac{Q(s)G(s)}{1+\left[G(s)-G_M(s)\right]Q(s)}r(s)}_{\text{Setpoint to output}} + \underbrace{\frac{1-Q(s)G_M(s)}{1+\left[G(s)-G_M(s)\right]Q(s)}d(s)}_{\text{Disturbance to output}} \quad (2)$$

It can be seen from this expression that perfect disturbance rejection is achieved by setting $Q(s)=G_M^{-1}(s)$. Furthermore, if $G(s)=G_M(s)$ (the model is exact), perfect set point tracking is also achieved.

3.2 Designing the IMC system

If the model is exact and minimum phase, the IMC design is trivial. However, this is usually not the case; consider a model with RHP zeros or a time delay where the inverse will lead to unstable and predictive (unrealizable) systems respectively. This is dealt with by factoring the model into an invertible and a non-invertible part as $G_M=G_+G_-$ and using only the invertible part G_-^{-1} as the controller.

A low pass filter $f(s)$ is typically introduced in series with the controller, motivation as follows. Using for the IMC controller

$$Q(s)=G_{-}^{-1}(s)f(s) \tag{3}$$

in Equation (2) and assuming no model mismatch gives:

$$y(s) = \underbrace{G_{+}(s)f(s)r(s)}_{\text{Setpoint to output}} + \underbrace{\left[1-G_{+}(s)f(s)\right]d(s)}_{\text{Disturbance to output}} \tag{4}$$

The filter is introduced for the following reasons:

- By choosing $f(s)$ in Equation (4) such that $G_{+}(0)f(0)=1$ eliminates both steady state error and disturbance.
- If $G(s)$ is strictly proper, its inverse will be improper. This can lead to very large excursions of the control signal and is not realizable in practice. Thus the order of $f(s)$ is chosen to make the controller proper.
- Filtering out high frequencies can reduce plant-model mismatch problems since the model is usually less accurate for high frequencies.

The simplest filter which has unity gain at steady state is

$$f(s) = \frac{1}{(\lambda s+1)^{r}} \tag{5}$$

where r is chosen to make the controller proper. The filter parameter λ is the only tuning parameter for the IMC controller and it should be set according to the bandwidth and robustness requirements.

3.3 Model mismatch issues

For the above statements, no model mismatch was assumed. This is naturally not the practical change since it is in most cases impossible to find a perfect model of a physical system. Of great importance is thus to ensure stability and performance even when model mismatch is present. If the set of possible true plants $G(s)$ is defined by the bounded multiplicative error $e_m < l_m$ as:

$$G(s) \in \left|G_{true}\cdot e_m\right| \tag{6}$$

where the relative error is given as:

$$e_m = \frac{G_{true}-G_m}{G_m} \tag{7}$$

The complimentary sensitivity function for the controller C is given as:

$$H = \frac{G_m C}{1+G_m C} \tag{8}$$

It can be shown [9] that the robust stability criterion for the closed loop system to be stable for all plants bounded by l_m is:

$$|H| < \frac{1}{l_m} \quad \forall \omega \tag{9}$$

3.4 IMC PID structure

For practical controller synthesis, the structure of Figure 3 is not the best choice since most commercially available controllers assumes a PID structure. To convert into the classical control structure, move the subtraction point of the plant / model output to before the $Q(s)$ as shown in Figure 2.

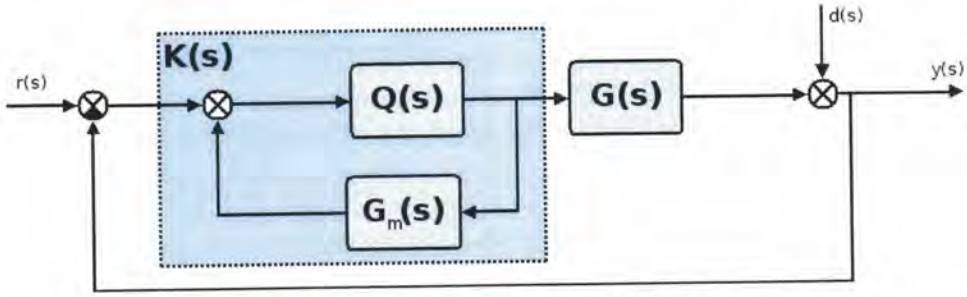

Figure 4: PID structure of Internal Model Control

Now the the IMC controller $Q(s)$ and the plant model $G_m(s)$ forms a positive feedback loop and can be merged together to the controller K(s):

$$K(s) = \frac{Q(s)}{1-Q(s)G_m(s)} \tag{10}$$

This is possible as long as $G_m(s)$ does not contain any delay elements, which would render a non-casual controller. The next section shows how this can be worked around by using Padé approximations for the dead time.

3.5 IMC structure for a first order with delay time

This section shows how to obtain the PID structure of a first order system with dead time. This is a common model in process industry and will also be used in this paper. The transfer function is given as:

$$G(s) = \frac{k}{\tau s+1} e^{-Ls} \tag{11}$$

the IMC controller is obtained by factoring out the time delay and using Equation (3):

$$Q(s) = G_{-}^{-1}(s) f(s) = \frac{\tau s+1}{k(\lambda s+1)} \tag{12}$$

The PID structure of this controller will be

$$K(s) = \frac{Q(s)}{1-Q(s)G_m(s)} = \frac{(1+\tau s)(2+Ls)}{k(2\lambda+L)s} \tag{13}$$

comparing this expression to the cascaded PID structure:

$$K_c + \frac{1}{T_i s} + \frac{T_d s}{T_f s+1}$$

gives the following parameters for the IMC-PID controller:

$$T_i = k(L+\lambda) \qquad K_c T_f + T_d = \frac{\tau L}{2k(L+\lambda)} \qquad K_c T_i + T_i + T_f = \frac{2\tau + L}{2} \tag{14}$$

Chapter 4: On-line Identification algorithm

To detect the varying time delay as well as the model parameter, an existing recursive algorithm is modified to suit the needs of the system. The algorithm is evaluated with MATLAB simulation.

Recursive delay estimation

For the IMC controller to be effective, it is necessary to have an accurate model is at hand. The key part in this system is the time delay which will change as the flow rate of the gaseous helium is changing. An accurate knowledge of the time delay is of utmost importance since a faulty assumption can make the system go unstable. To improve performance, an on-line algorithm is presented in this chapter. Among the properties is its ability to effectively estimate a changing time delay in a recursive fashion.

Original delay estimating algorithm

An attractive method for estimating time delays that can be attached to an existing recursive estimator algorithm is presented in [10]. The algorithm is an modification of any standard recursive parameter estimation algorithm and can easily be applied without changing anything in the original scheme. Compared to other algorithms which estimates the dead-time by a polynomial, this algorithm will not increase the number of parameters of the model and thus will be more computational efficient. In this section, the initial form of the algorithm is presented. subsequent chapters shows how the algorithm was modified to increase performance.

4.1 The algorithm

Using the ARMAX model, the output of the system is given as:

$$y(t) = z(t)\theta + e \tag{15}$$

where

$$z(t,d) = [-y(t-1),\ldots,-y(t-n),\ u(t-d-1),\ldots,u(t-d-m)] \tag{16}$$

is the vector of input and measured output values and

$$\theta = [a_1, a_2, \ldots, a_n, b_1, b_2, \ldots, b_m]^T \tag{17}$$

is the parameter vector. d is the delay time of the system given in number of samples. The predicted output is given as:

$$\hat{y} = \hat{z}(t)\hat{\theta} \tag{18}$$

The prediction error is defined as:

$$\epsilon(t) = y(t)-\hat{y}(t) = y(t)-\hat{z}(t)\hat{\theta} \tag{19}$$

In every step of the recursive identification algorithm, the accumulating performance index at time t is calculated as:

$$J = \sum_{i=t}^{i=0} \epsilon(t-i)^2 \tag{20}$$

The idea of the method is to minimize (20) with respect to the delay time to find out which delay time gives the smallest error. To accomplish this, J is evaluated for all the delays given by the interval $[d_{min}, d_{max}]$ and the delay producing the smallest value of J is taken as the predicted delay, $\hat{d}$. The predicted delay is then assumed to be the correct one in the next step of the iteration when finding $z(t)$ in Equation (16). Now the parameters will be estimated a little bit better (since the delay is a little bit better estimated) and the next delay will in turn be estimated better since the parameters are a little bit better estimated. Thus the two are pulling each other up in a bootstrap manner.

Implemented delay detection algorithm

Now the algorithm is to be implemented for the temperature process described by Equation (11). Discretization with sample period T_s gives the following representation in the Z-domain:

$$G_{disc}(z) = \frac{b}{z+a}\cdot z^{-d} \tag{21}$$

where the discrete delay $d = L/T_s$ We have $\theta = [a,\ b]^T$ and $z(t) = [-y(t-1),\ u(t-1-d)]^T$.

First consider the case of constant parameter estimation of θ with the recursive ARMAX estimator along with the delay estimation method presented earlier. The algorithm will look as follows:

Z vector with estimated delay: $z(t) = [-y(t-1),\ u(t-1-\hat{d})]^T$

Error:
$$\epsilon = y(t)-z^T(t)\hat{\theta}(t-1)$$
$$\zeta = z^T(t)P(t-1)z(t)$$

Error covariance:
$$P(t) = P(t-1)-\frac{P(t-1)z(t)z^T(t)P(t-1)}{1+\zeta(t)}$$

Estimated parameters:
$$\hat{\theta}(t) = \hat{\theta}(t-1)+\frac{P(t-1)z(t)}{1+\zeta(t)}\epsilon(t|t-1) \tag{22}$$

Performance index array:
$$J = \sum_{i=t}^{i=0} \epsilon(t-i)^2 \quad \forall d\in[d_{min}, d_{max}]$$

Find minimum value of J:
$$J_m = min\, J(k,d) \quad \forall d\in[d_{min}, d_{max}]$$

Estimated delay:
$$\hat{d} = m+d_{min} \qquad (m \text{ is the index of } J_m)$$

m in the last equation is the index of the lowest value of J.

These equations are iterated for every sampling point. They require the specification of an initial parameter vector θ_0 along with an initial error covariance estimation P_0 which is based on how accurate

θ_0 is believed to be. The initial parameters can be obtained from measurement data.

Improvements of the algorithm

Some improvements made to the algorithm are presented here. The resulting flow chart describing the algorithm can be seen in Figure 5.

4.1 Adaptive delay range

Since the time delay will relatively large, the number of input values stored for computation of J will also be large. An adaptive delay range is introduced here which will change the number of stored values depending on the estimated delay time. An nominal guess of the delay, d_0, can be obtained from the measurement of helium flow rate together with information about the total length of the pipes. The nominal delay will decide the range of possible delays, $[d_{min}, d_{max}]$, which has to be changed accordingly if the flow rate is changed. The proposed algorithm changes these limits automatically to keep the current estimated delay at the center of the interval. Thus d_{min} and d_{max} is calculated as:

$$d_{min} = \hat{d} - d_\Delta \qquad d_{max} = \hat{d} + d_\Delta \tag{23}$$

d_Δ is a tuning parameter which will decide the range of delays. The algorithm will be able to track the delay as long as the delay is not changed with more than d_Δ between two samples.

4.2 Summation horizon

The array J consists of the sum of all squared errors for different delay times. Since the error decreases as the parameters are estimated more accurately, the added terms are getting smaller and smaller and will ideally approach zero if there is no noise present. Consider a time delay of d_1 samples. After a given time, the minimum of J will converge to the cell representing d_1. Now if the delay time changes to d_2, the cell representing d_2 will receive the minimum value added in every cycle. However it will take a long time for the d_2 cell to become the new minimum because of the value already present in d_1. To prevent this and allow for changing delay time, the summation horizon when calculating J is changed from zero to $t-T_j$ to only account for the T_j+1 most recent samples. The new performance index will look like:

$$J = \sum_{i=t}^{t-T_j} \epsilon(t-i)^2 \quad \forall\, d \in [d_{min}, d_{max}] \tag{24}$$

Thus the new value of J will not contain any information about measurements made before $t-T_j$.

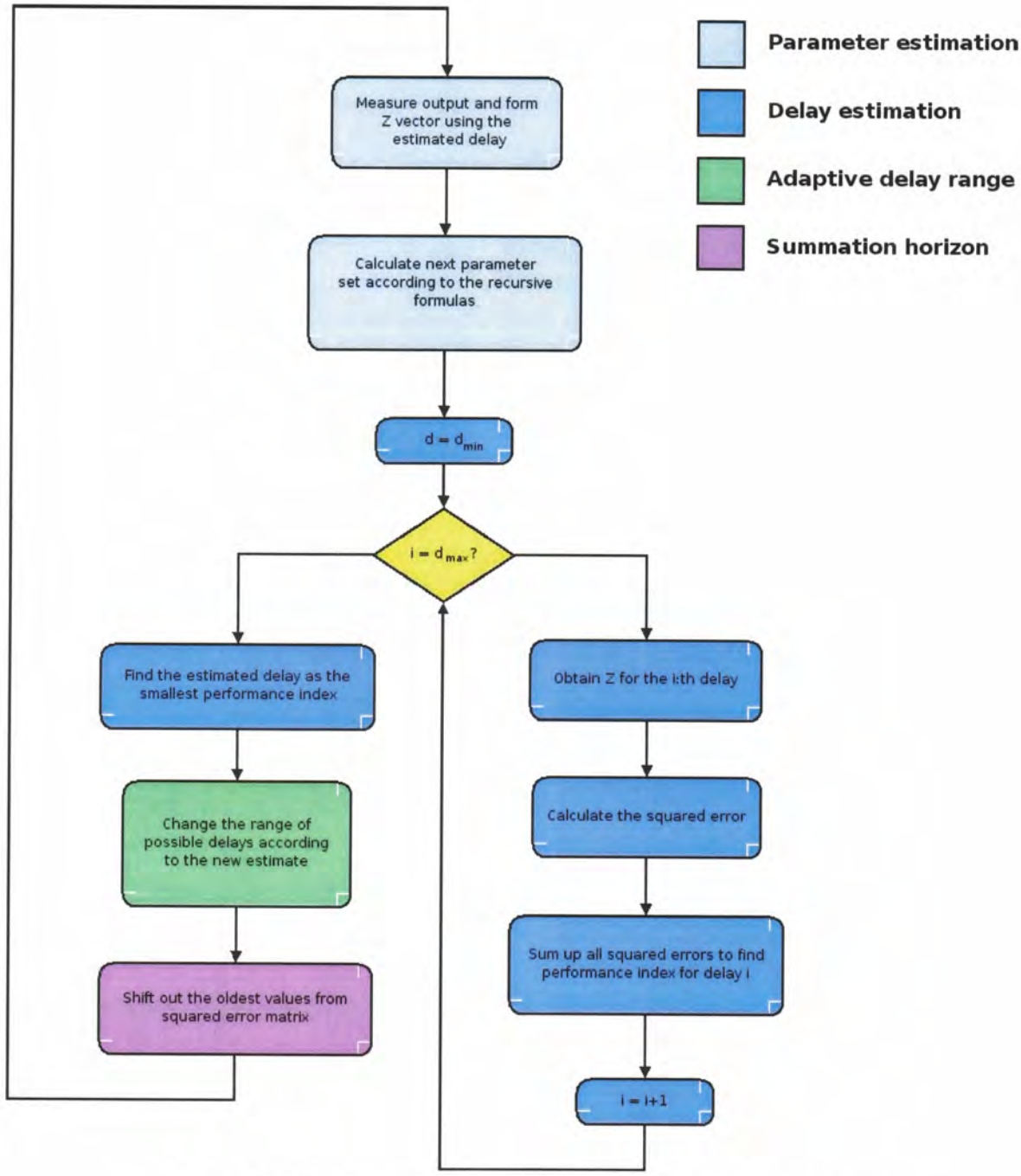

Figure 5: Flowchart of recursive parameter detection algorithm

Simulation

The algorithm was simulated in MATLAB, the code can be seen in Appendix A.

4.1 Settings

The simulated plant is given as the discrete transfer function:

$$G(z) = \frac{0.09516}{z-0.9048} z^{delay} \tag{25}$$

The delay is initially given as 50 samples and is changed to 46 after half of the simulation time has passed. A Gaussian noise with a standard deviation of 0.01 is applied to the measurement values. d_{min} and d_{max} are initially set so that the true delay time is in the middle of the interval which size is given by $d_{\Delta} = 50$.

4.2 Results

The results from the simulation is presented in the following plots. The change of delay time is applied at $t = 250$ and it can be seen from Figure 6 that the parameters estimates are exhibiting a small change at this time. The difference between true and estimated output (Figure 7) is seen to be very large at this time but since the parameters are already "tuned in", they are not able to change that rapidly. The estimated delay can be seen in Figure 8 and its ability to track the change of delay with relatively small glitches is evident.

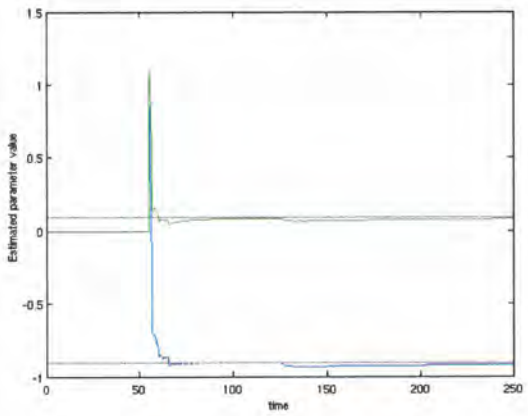

Figure 6: Estimated parameter values and their corresponding true values

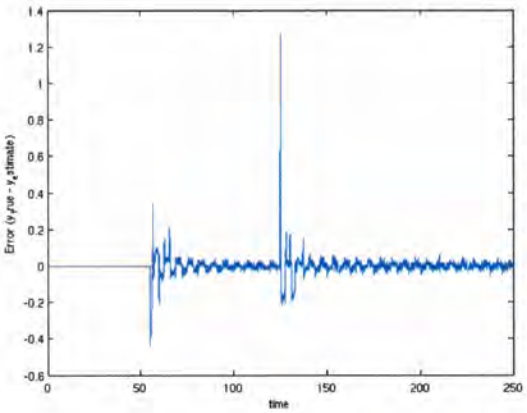

Figure 7: Difference between measured and estimate value (error)

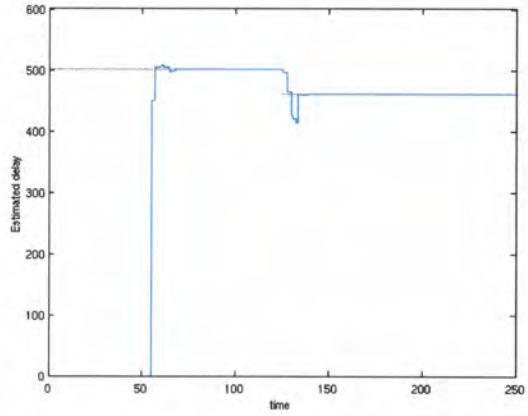

Figure 8: Estimated delay time

Chapter 5: Proposed Control Structure

This chapter firstly presents the model used for the 300K-80K cooling system of the AMS-02 magnet. In the subsequent section, the proposed control structure can be found.

Model of the 300K-80K cooling system

The control aspects of the 300K-80K cooling system can be seen in Figure 9. Valves V_1 and V_2 are to be used for regulating the temperature of the output flow T_{out}. V_1 controls the cool flow of 77K, while V_2 controls the intermediate temperated flow (see Figure 9). The actuators have some inherent dynamics $G_a(s)$.

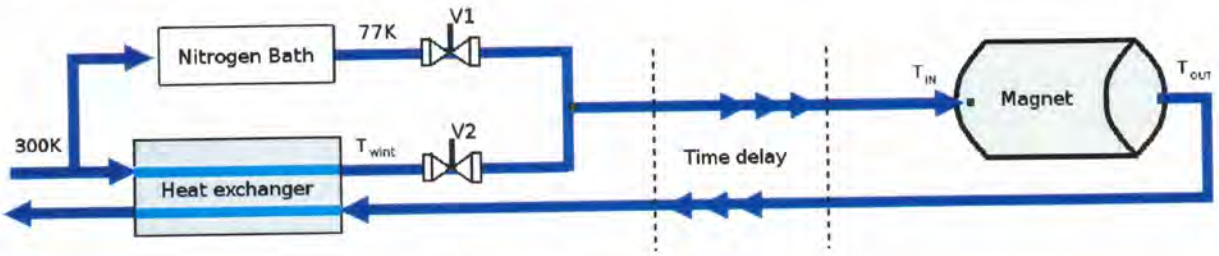

Figure 9: Control aspects of 300K-80K cooling system

There are two sources of time delays in the structure, the first one being the transport delay L_1 in the cryogenic pipes from the cooling system to the magnet as indicated in the figure. The second one is the pipes inside the magnet itself, which is modeled here as a pure time delay L_2. The output temperature will be slightly reduced from the heat exchange between the magnet coils and the flow. However the time delay will be of a much lesser magnitude as compared to the thermal absorption rate of the magnet (the whole cooling process is estimated to take 14 days) so the magnet is here modeled as a pure delay with a disturbance. The loop transfer function $G(s)$ will thus consist of a delay element of time $L = L_1 + L_1$ and non-delayed transfer function $G_0(s)$ which will mainly be determined by the actuator dynamics. There is a sensor at the input of the magnet so that the temperature between the two delays can be measured.

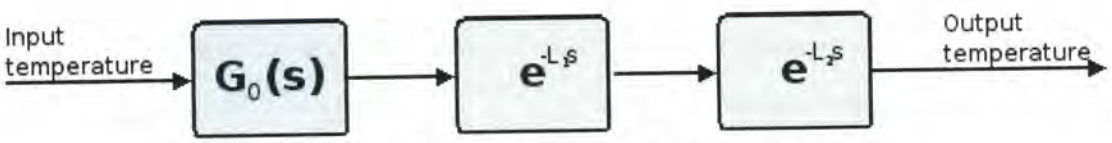

Figure 10: Block diagram of the model

Since there AMS-02 mechanical system is not yet finished, there are no test data available for modeling.

Here it is assumed that the transfer function can be given as a first order system with delay:

$$G(s) = \frac{k}{\tau s+1} e^{-(L_1+L_2)s} \tag{26}$$

τ is here assumed to be of the same order as the total time delay L. The gain k represents thermal losses during the flow. Sources of random nature such as temperature fluctuations at the output due to unmodeled dynamics of the pipe system and sensor noise is accounted for as a Gaussian disturbance at the output.

The presented control problem boils down to the control of a first order system with a time delay. The key part here is that the time delay is large as compared to the system time constant and will thus have great inflict on the maximum allowed bandwidth of the system.

Internal model control structure

The Internal Model Control schematics for the system is presented in Figure 11.

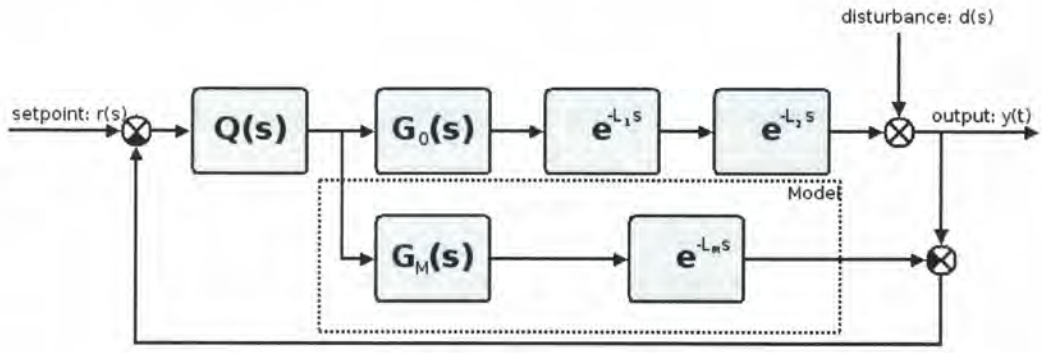

Figure 11: Basic IMC structure of cooling system

$G_M(s)$ is here representing the modeled non-delayed transfer function and L_M is the delay in the model. Now the IMC controller can be determined using the results from Chapter 3. Since the minimum phase part of the model is a first order system, the controller will simply be its inverse together with a first order low pass filter:

$$Q(s) = \frac{\tau s+1}{k(\lambda s+1)} \tag{27}$$

Adaptive IMC

As the plant's parameters are changing, the controller and the model has to change accordingly. The identification method presented in Chapter 4 is used here to form an adaptive controller which can account for changes in time delay.

Considering the discrete case, the adaptive IMC controller will be given as the inverse of the minimum phase part of (21) together with a discrete IIR filter, defined by the parameter λ_d:

$$Q(z) = \frac{z+\hat{a}}{\hat{b}(\lambda_d z+1)} = \frac{y(z)}{r(z)} \tag{28}$$

where $\hat{a}$ and $\hat{b}$ are the estimated parameters from Equation (22). Rewriting (28) on filter form:

$$u(z) = \frac{1}{\hat{b}\lambda_d}\left[r(z) + r(z)\hat{a}z^{-1} - y(z)\hat{b}z^{-1}\right] \tag{29}$$

gives a controller ready for implementation. The same is performed for the model. Using the estimated delay $\hat{d}$ gives for the model:

$$y_M(z) = \hat{b}u(z)z^{-1-\hat{d}} - \hat{a}y(z)z^{-1} \tag{30}$$

Chapter 6: Real-time simulation

Introduction

To evaluate the real time performance of the suggested control system, a simulation program was constructed using BORLAND VISUAL STUDIO C++. The program implements the true plant as a discrete system with a random disturbance and the modeled plant the same discrete system but without the disturbance. The previously presented identification algorithm is used to deduce the true plant's parameters and subsequently decide the parameters of the controller as given in Equation (29). The C++ code for this can be seen in APPENDIX B.

Sub-parts

6.1 Signals class

Since previous values of the signals has to be kept in memory, a class was created for storage of the signal. It works essentially as an array that shifts out the old value every time a new value is added with its *Append* - method.

6.2 LTI_system class

This class is used for the rational part of the controller, the true plant and the model. It works basically as a filter, where the filter coefficients are given upon creation of the object. These coefficients can thereafter be changed during operation (for the adaptive controller case). The method *output* takes an input and output array as arguments and calculates the system's output by convolution.

6.3 Time delay class

The time delay class creates a time delay object with a specified dead-time. It works in a similar fashion as the *Signal* class. When a value is appended t the input of the time delay, all current values are shifted one sample. The output is read from the end of the array.

6.4 Disturbance

The disturbance is generated as a pseudo-random sequence using the *rand()* function. This will not produce a truly random sequence but will satisfy in this case.

6.5 Estimator

The parameter and time delay estimator is implemented as an own class which uses the results from Chapter 4.

6.6 Timer

The system layout from Figure 11 is implemented, with the difference that only one total delay for the true plant is used. With a sampling period of 10ms, a *Timer* us used to execute a portion of code every 10:th ms. In this code, all outputs form the controller or plants are calculated and the signal vectors are updated. The estimation algorithm is also progressed one step. Finally, the new results are plotted to the chart and labels are updated.

Operation

There are various parameters the user can change on-line during the operation of the program:

- Reference input can be set in the range between 0 and 1.
- The delay of the true plant can be changed.
- Magnitude of the disturbance.
- Adaptive controller on/off. Decides whether the model will use the estimated or the true time delay.

Other settings such as plant parameters and sampling time can be set directly in the code. The measurements are presented in the graph window where the following measurements are available:

- Output of the system.
- Output from the controller.
- Error between estimated and true output.
- Estimated and true delay time.

Simulation results

For the following simulations, the discrete plant

$$y(t) = 0.4877u(t-d)-0.9512\mathrm{y}(t) \tag{31}$$

is used. The same relation is used for the model, the only difference being the delay which will change according to the estimated value if the adaptive option is enabled. For the estimator, it is assumed that the plant rational parameters are know exactly, and thus the only parameter being estimated is the time delay. The corresponding IMC controller output is given as

$$u(t) = -0.019\mathrm{u}(t-d)+0.02u(t-d-1)-0.99\mathrm{y}(t) \tag{32}$$

The results from a step input can be seen in Figure 12 and delay estimating performance can be seen in Figure 13. The speed of the step response can be changed by redesigning the controller. The delay estimator works well as long as there is some action on the inputs. if the input is unchanged for a time longer than the specified horizon, the value is lost.

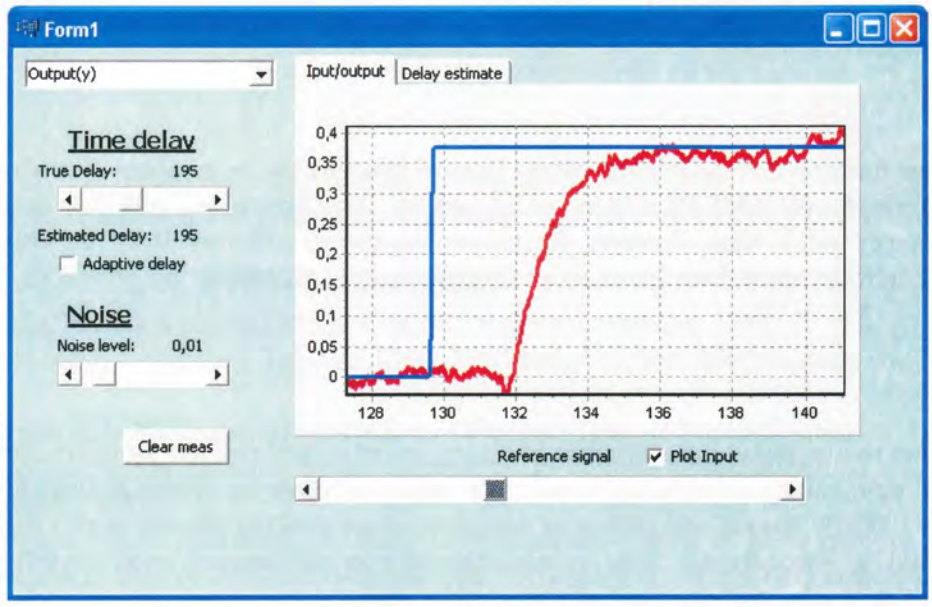

Figure 12: Result from a step input. Blue plot is reference and red is output

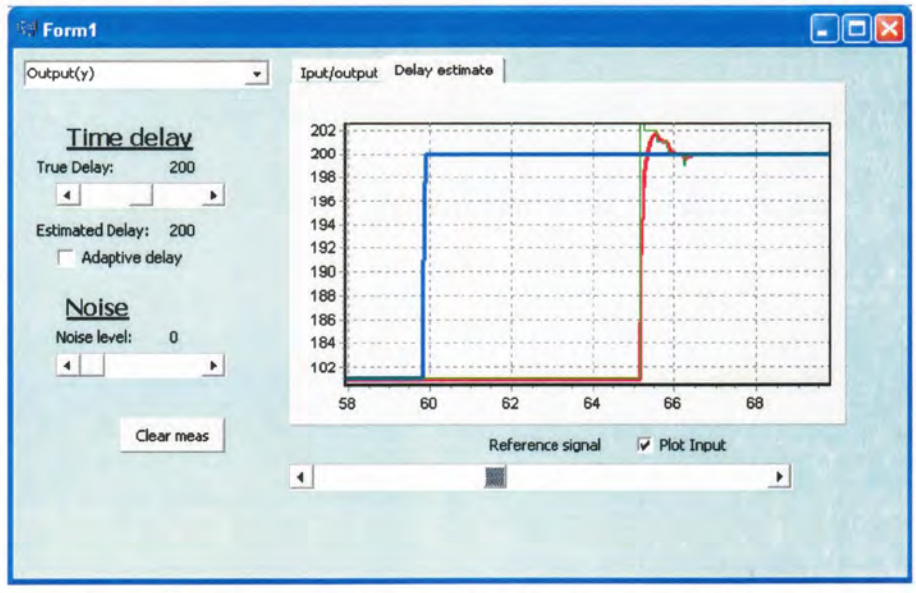

Figure 13: Change of true delay (blue) and estimated delay (red)

České vysoké učení technické v Praze
FAKULTA ELEKTROTECHNICKÁ
Katedra řídicí techniky
121 35 Praha 2, Karlovo nám.13
Tel.: 224922372, Fax: 224918646

Chapter 7: Conclusion

Finally we can mention that the Internal Model Control structure can be an alternative when considering control mechanism for the AMS-02 magnet cooling system. Its ability to deal with time delay and disturbances comes well at hand. However, the real cooling system will most likely run with a standard (proven) PID controller since there are no room for experimental excursions in a project of such importance.

Further work

The simulation results shows the usefulness of the suggested control method. However there are some glitches in the detection algorithm which needs to be improved before the system is ready for practical implementation. Firstly, the rational parameter estimation is not working properly at this stage and this has to be worked on. Secondly, the delay estimation shows poor performance under the influence from disturbance which also has to be worked around in some way.

The initial intention was to implement the control system on the SIEMENS S7-400 PLC for practical test in a heater/valve configuration. This is the same control network system as the real AMS-02 cooling system will be implemented with the same temperature sensors, valve, etc. The further work includes refining of the detection algorithm and then implementation into this system. There is also intentions of creating a IMC-PID system for comparison with the standard IMC structure and maybe implement this structure practically if it simulations are proved useful.

References

1: B. Blau et al., The Superconducting Magnet System of AMS-02, 2002

2: Carlos E. Garcia and Manfred Morari, Internal Model Control 1. A Unifying Review and some new Results, 1982

3: , Internal Model Control. 4. PID Controller design,

4: Kou Yamada, Modified Internal Model Control for unstable systems, 1999

5: Wen Tan, Horacio J. Marquez, Tongwen Chen, IMC design for unstable processes with time delay, 2002

6: Rui Wang and Keiji Watanabe, Three-Degree of Freedom Internal Model Control, 2006

7: Muhammad Shafiq, Sayyid Hasan Riyaz, Internal Model Control Structure Using Adaptive Inverse Control Strategy, 2003

8: Keiji Watanabe, Eiichi Muramatso, Adaptive Internal Model Control of SISO systems, 2003

9: Sigurd Skogestad, Ian Postlethwaite, Multivariable Feedback Control, analysis and design, second edition, 2005

10: Ashraf Elnaggar, Guy A. Dumont and Abdel-Latif Elshafei, Recursive estimation for systems of unknown delay, 1989

Appendix A: Identification Algorithm with MATLAB

```
clear all;

%Simulation settings
Ts = 0.1;                   %Sampling period
t_sim = 250;                %Simulation time in seconds
N = ceil(t_sim / Ts);       %Number of sample points
t = [0:Ts:t_sim-1/t_sim]; %Time vector

%ARMAX model (currently only working for n_b = n_a = 1)
n_b = 1;    %no of parameters in numerator
n_a = 1;    %no of parameters in denominator
n_p = n_b + n_a;

%Plant parameters
k = 1;          %DC gain
tau = 1;        %Time constant
L = 50;         %Delay1
L2 = 46;        %Delay2
Y_stdev = 0.01;%Measurement noise st. deiation

%Plant setup
%Two plants with different time delays are used
G = tf([k],[tau, 1] , 'inputdelay', L); %first plant
G_disc = c2d(G,Ts);
[num, den] = tfdata(G_disc, 'v');
G_disc2 = tf([0 num(2)*1],[den], Ts,'inputdelay', L2/Ts);   %Second plant

%Generate signals
r = square(t);              %input is a square wave
y_true = lsim(G_disc,r);    %output from system 1
y_true2 = lsim(G_disc2,r);  %output from system 2
y_meas = [y_true(1:N/2); y_true2(N/2:N-1)]   + randn(size(y_true))*Y_stdev;
          %first half from first system, second half from second system

%Initialize recursion algorithm
    %Initial range of the estimated delay
    d_diff = 50;
    d_min = L/Ts -d_diff;
    d_max = L/Ts +d_diff;

    THETA_old = zeros( (n_p), 1) + 1;   %Initial parameter vector
    THETA_values = zeros((n_p),N);      %Parameters stored for later plot
    P_old = 10*eye(n_p);                %Initial error covariance
    d_hat = (d_min+ d_max) / 2;         %Initial delay estimate
    J_old = 0*[d_min : d_max-1];        %Performance index array
    d_stored = zeros(1,N);              %Delays stored for later plot
    E_size = 50;                        %Number of stored error values
    E_a = zeros(E_size, size(J_old,2) );%Matrix of stored errors
```

```
%PARAMETER ESTIMATION
for i= 1+n_a+d_max : N;

%RECURSIVE ARMAX
   %Create the Z-vector using estimated delay
        Z = [ -y_meas( (i-n_b):(i-1) ); r( (i-d_hat-n_b):(i-d_hat-1) )'   ] ;

   %Calculate error
        error = y_meas(i) - Z'*THETA_old;
        error_values(i) = error;
   %Update error covariance
        zeta = Z'*P_old*Z;
        P_new = P_old - (P_old*Z*Z'*P_old)   /   (1+zeta);

   %Update estimated parameters
        THETA_new = THETA_old + error * (P_new*Z)   /   (1+zeta);
        THETA_values(:,i) = THETA_new;

        P_old = P_new;
        THETA_old = THETA_new;

%DELAY ESTIMATION
    %Shift out the oldest error
        E_a = circshift(E_a,1);

    %Calculate performance index for all delays in the current range
        j=1;    %index counter
   for d = d_min : d_max-1
     %Get Z for delay d
       Z_d = [ -y_meas( (i-n_b):(i-1) ); r( (i-d-n_b):(i-d-1) )'   ];
     %Calculate squared error
       E_a(1,j) = (y_meas(i) - Z_d'*THETA_new)^2;
     %Sum to form the performance index
       J(j) = sum(E_a(2:E_size,j));
       j=j+1;
   end

    %Find the estimated delay as the minimum performance index
        [J_min, index] = min(J);
        d_hat = d_min-1 + index;
        d_stored(i) = d_hat;

    %Determine the new delay range
    if mod(i,50) == 0

        if i > d_max+1

            d_min = round(d_hat - d_diff);
            if d_min < 1
               d_min = 1;
            end
            d_max = d_min + 2*d_diff;
        end
    end
end
```

```
%end of parameter estimation

%Plot the results
figure(1)
hold off;
plot (t, THETA_values)
hold on;
plot (t, num(2));
plot (t, den(2));
xlabel('time')
ylabel('Estimated parameter value')

figure(2)
hold off;
plot (t(1:N/2),L/Ts)    %plot true delay
hold on;
plot (t(N/2:N-1),L2/Ts)
plot(t,d_stored)
xlabel('time')
ylabel('Estimated delay')

figure(3)
plot(t,error_values);
xlabel('time')
ylabel('Error (y_true - y_estimate)')
```

Appendix B: Code for VISUAL C++ Simulation Application

Definitions

```
#define TS 0.01                 //Sampling period

//Model parameters
//Models are given on the form:
//                    b1 + b2*u^z + b3*u z^2 + ...
//G(z) =    ------------------------------
//                1 + a1 z + a2 z^2 + ...
#define MODEL_A_SIZE 1
#define MODEL_B_SIZE 1
#define MODEL_a1 -0.99
#define MODEL_b1 0.00995
#define MODEL_A_ARRAY {MODEL_a1}
#define MODEL_B_ARRAY {MODEL_b1}
#define Y_STDEV 0.01

//Controller parameters
#define LAMBDA 10
#define CNTRL_A_SIZE 1
#define CNTRL_B_SIZE 2
#define CNTRL_A_ARRAY {- 0.995} //{-0.99}
#define CNTRL_B_ARRAY  {- 0.495, 0.5}//{-0.019, 0.02}
//#define CNTRL_A_ARRAY {-0.09516}
//#define CNTRL_B_ARRAY {0.9048}


//Time delays given in no of samples
#define T1_DEADTIME 200
#define T2_DEADTIME 200

//Estimator
#define DELAY_DIFF 20

#define E_SIZE 1000
#define INPUT_SCALING 1000 //

#define STANDARD_SIGNAL_SIZE 3
#define OUTPUT_SIZE 3
#define REF_SIZE 3
```

Parameter Estimator

```
class Parameter_estimator
{
	public:
	float err;
	Matrix *THETAarray;
	float *Ahat, *Bhat, *J;
	int Asize, Bsize, noParam, Jsize, dmin, dmax, dhat;
	Matrix Z, Zd, Zdold, P, THETA, error, zeta, meas, measold;

	//Constructor
	Parameter_estimator(int bLength, int aLength)
	{
	dhat = T1_DEADTIME;
		Asize = aLength;
		Bsize = bLength;
		noParam = Asize+Bsize;
		dmax = T1_DEADTIME + DELAY_DIFF;
		dmin = T1_DEADTIME - DELAY_DIFF;
		Jsize = DELAY_DIFF*2;

		dhat = T1_DEADTIME;

		THETAarray = new Matrix[E_SIZE];
		J = new float[Jsize];
		Ahat = new float[Asize];
		Bhat = new float[Bsize];
		err = 0;
		P.SetSize(2,2);  //error covariance matrix
		P(0,0)=1;  P(1,1)=1;   P(0,1)=0;  P(1,0) = 0;
		Z.SetSize(2,1);
		Z(0,0) = 0; Z(1,0) = 0;
		Zd.SetSize(2,1);
		Zd(0,0) = 0; Zd(1,0) = 0;
		Zdold.SetSize(2,1);
		Zdold(0,0) = 0; Zd(1,0) = 0;
		error.SetSize(1,1);
		error (0,0) = 0;
		zeta.SetSize(1,1);
		zeta(0,0) = 0;
		THETA.SetSize(2,1);
		THETA(0,0) = MODEL_b1; THETA(1,0) = MODEL_a1;
		meas.SetSize(1,1);
		measold.SetSize(1,1);
		for (int i=0; i < E_SIZE; i++)
		{
			THETAarray[i].SetSize(2,1);
			THETAarray[i](0,0) = 0; THETAarray[i](1,0) = 0;
		}
		for (int i=0; i < Asize; i++)
		{
			Ahat[i] = 0;
		}
```

```
            for (int i=0; i < Bsize; i++)
      {
            Bhat[i] = 0;
      }
            for (int i=0; i < Jsize; i++)
      {
            J[i] = 0;
      }
}


//Method for updating the rational parameter estimation
void update(float u, float y)
{
      meas(0,0) = y;
      Z(0,0) = u;
      Z(1,0) = -y;

      //Calculate error
      error =  meas - (~Z*THETA);
      //Update error covariance
      zeta = ~Z*P*Z;
      zeta(0,0) = zeta(0,0) + 1;
      P = P - (P*Z*~Z*P)  / (zeta(0,0));
      THETA = THETA + error(0,0) * (P*Z) / (zeta(0,0));

      appendValue(THETA);                  //Add value to the THETA array
}
//Method for estimating the delay
void delayEstimate(float *u_ptr, float *y_ptr)

{
      int j = 0;
//Calculate the performance index for every loop in the range
      for (int d=dmin; d < dmax-1; d++)
      {
            Zd(1,0) = -*(y_ptr + 1);
            Zd(0,0) = *(u_ptr + d + 1);

            Zdold(1,0) = -*(y_ptr + 1 + E_SIZE);
            Zdold(0,0) = *(u_ptr + d + 1 + E_SIZE);

            meas(0,0) = *y_ptr;
            measold(0,0) = *(y_ptr + E_SIZE);

            J[j] = J[j] + pow( (meas - ~Zd*THETA)(0,0),2)
                        - pow( (measold - ~Zdold*THETA)(0,0),2);
      j++;
      }
      //Find the smallest J

      float smallest = J[0];
      int index = 0;
      for (int i=1; i < Jsize-1; i++)
      {
```

```
                if (J[i] < smallest)
                {
                        smallest = J[i];
                        index = i;
                }
        }
        dhat = dmin + index + 1;
        err = *y_ptr - (THETA(0,0)* *(u_ptr+1+dhat) - THETA(1,0)* *(y_ptr+1)  );
    }

    void appendValue(Matrix value)
    {
        int i;
        for (i = (E_SIZE-1); i>0; i--)
        {
            THETAarray[i] = THETAarray[i-1];    //Shift out the values
        }
        THETAarray[0] = value;
    }
};
```

Time Delay

```
class Time_delay
{
public:
    float *array;
    int delay;

    //Constructor
    Time_delay(int delay_size)
    {
        delay = delay_size;
        array = new float[delay_size];
        for (int i=0; i < delay_size; i++)
        {
        array[i] = 0;     //initialize array
        }
    }


    //Append value
    void appendValue(float value)
    {
        int i;
        for (i = (delay-1); i>0; i--)
        {
            array[i] = array[i-1];  //Shift the values
        }
        array[0] = value;
    }
    float readValue()
    {
    //read the last value
```

```
            return array[delay-1];

      }

};
```

Signal Vector

```
class Signal_vector
{
    public:
      float *array;     //the array
      int size;


      Signal_vector(noOfValues)
      {
            size = noOfValues;
            array = new float[size];
            int i;
            for (i = 0; i<size; i++)
            {
                  array[i] = 0;     //Initialize array
            }
      }

      //Append a value to the beginning and shift all other values
      void appendValue(float value)
      {
            int i;
            for (i = (size-1); i>0; i--)
            {
                  array[i] = array[i-1];  //Shift out the values
            }
            array[0] = value;
      }
      float readValue(int index)
      {
            if (index > size-1)
            {
            return 0;
            }

      return array[index];
      }

};
```

LTI system

```
class LTI_system
{
      public:
```

```
    int B_size, A_size;

    float *B;   //numerator coefficients
    float *A;   //denominator coefficients

    LTI_system(int numSize, int denSize, float *B_ptr, float *A_ptr)
    {
        B_size = numSize;
        A_size = denSize;

        B = new float[B_size];
        A = new float[A_size];

        for (int i=0; i < B_size; i++)
        {
            B[i] = *B_ptr;
            B_ptr++;
        }
        for ( int i=0; i < A_size; i++)
        {
            A[i] = *A_ptr;
            A_ptr++;
        }

    }

    float calculateOutput(float *u_ptr, float *y_ptr)
    {
        float output = 0;
        for (int i=0; i < B_size; i++)
        {
            u_ptr++;
            output = output + B[i] * *u_ptr;
        }
        for (int i=0; i < A_size; i++)
        {
            u_ptr++;
            output = output - A[i] * *y_ptr;
        }

        return output;
    }
};
```

Create objects (executed at program startup)

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
float t = 0; //time


 //Create signals
ref = new Signal_vector(REF_SIZE);
q_in = new Signal_vector(STANDARD_SIGNAL_SIZE);
q_out = new Signal_vector(T1_DEADTIME + DELAY_DIFF + E_SIZE);
```

```
d_out = new Signal_vector(STANDARD_SIGNAL_SIZE);
dm_out = new Signal_vector(STANDARD_SIGNAL_SIZE);
gm_out = new Signal_vector(STANDARD_SIGNAL_SIZE);
feedback = new Signal_vector(STANDARD_SIGNAL_SIZE);
disturbance = new Signal_vector(STANDARD_SIGNAL_SIZE);
y = new Signal_vector(T1_DEADTIME + DELAY_DIFF + E_SIZE);

//Create models and time delay
T1 = new Time_delay(T1_DEADTIME);
T2 = new Time_delay(T2_DEADTIME);

float A_m[MODEL_A_SIZE] = MODEL_A_ARRAY;
float B_m[MODEL_B_SIZE] = MODEL_B_ARRAY;
G = new LTI_system(MODEL_B_SIZE, MODEL_A_SIZE, B_m, A_m);
Gm = new LTI_system(MODEL_B_SIZE, MODEL_A_SIZE, B_m, A_m);

float A_d[1] = {- 0.9048};
float B_d[1] = {0.09516};
D_filter = new LTI_system(1, 1, B_m, A_m);

float A_c[CNTRL_A_SIZE] = CNTRL_A_ARRAY;
float B_c[CNTRL_B_SIZE] = CNTRL_B_ARRAY;
Q = new LTI_system(CNTRL_B_SIZE,CNTRL_A_SIZE,B_c,A_c);

//Estimator
est = new Parameter_estimator(1,1);


      srand(time(0));  // Initialize random number generator.

      ScrollBar2->Min = T1_DEADTIME - DELAY_DIFF;
      ScrollBar2->Max = T1_DEADTIME + DELAY_DIFF;
      ScrollBar2->Position = T1_DEADTIME;

float d_est = 0;
}
```

Timer function (executed every sampling period)

```
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
      float x = 0;

      //disturbance
      x = (0.1*(rand() % 10) - 0.5) * ScrollBar3->Position / 10000;
      disturbance->appendValue(x);

      //Signal Ref (input signal)
      x = (float)ScrollBar1->Position / INPUT_SCALING;
      ref->appendValue(x);

      //Input to controller: q_in = ref - feedback
      x = ref->readValue(0) - feedback->readValue(0);
      q_in->appendValue(x);
```


České vysoké učení technické v Praze
FAKULTA ELEKTROTECHNICKÁ
Katedra řídicí techniky
121 35 Praha 2, Karlovo nám.13
Tel.: 224922372, Fax: 224918646

```
        //Output of controller: q_out = q_in conv Q
        x = Q->calculateOutput(q_in->array, q_out->array);
        q_out->appendValue(x);

        //output from true delay: d_out(t) = q_out(t-d)
        x = q_out->readValue(0);
        T1->appendValue(x);
        x = T1->readValue();    //get the output from delay
        d_out->appendValue(x);

        //signal y
        x = G->calculateOutput(d_out->array, y->array);
        y->appendValue(x + disturbance->readValue(0));

        //signal gm_out
        x = Gm->calculateOutput(dm_out->array, gm_out->array);
        gm_out->appendValue(x);

        //output from modeled delay: dm_out(t) = q_out(t-d)
        x = q_out->readValue(0);
        T2->appendValue(x);
        x = T2->readValue();    //get the output from delay
        dm_out->appendValue(x);

        //signal feedback
        x = y->readValue(0) - gm_out->readValue(0);
        feedback->appendValue(x);

        //filter the estimated delay

        d_est = est->dhat*0.09516 + d_est*0.9048;


              Label13->Caption = d_est;

         //Adaptive control
         if (CheckBox2->Checked == true)
         {


              T2->delay = est->dhat;
}
         else
         {
              T2->delay = ScrollBar2->Position;
    }
         Label5->Caption = ScrollBar2->Position;
         T1->delay = ScrollBar2->Position;


         Label1->Caption = est->THETA(0,0);
         Label2->Caption = est->THETA(1,0);
         Label4->Caption = est->dhat;
         Label6->Caption = t/TS;
         Label11->Caption = (float)ScrollBar3->Position / 10000;
```

```
        if (TabControl1->TabIndex == 0)
        {
              if (ComboBox1->ItemIndex == 0)
              {
                    Series1->AddXY(t,y->readValue(0));
              }
              if (ComboBox1->ItemIndex == 1)
              {
                    Series1->AddXY(t,q_out->readValue(0));
              }
              if (ComboBox1->ItemIndex == 2)
              {
                    Series1->AddXY(t,est->err);
              }
              if (CheckBox1->Checked == true)
              {
              Series2->AddXY(t,ref->readValue(0));
              }
        }
        if (TabControl1->TabIndex == 1)
        {
              Series2->AddXY(t,ScrollBar2->Position);
              Series3->AddXY(t,est->dhat);
              Series1->AddXY(t,d_est);
    }
        //update time
        t = t + TS;
}
```